

**Relatório do Trabalho de Algoritmos e Estruturas de Dados**

*Gestão de Bilhetes no Metro – Parte II*

**2019**

por

João Pedro Ribeiro | 201704851 | up201704851@fe.up.pt

Rui Teixeira | 201707001 | up201707001@fe.up.pt

Vítor Ventuzelos |201706403 | up201706403@fe.up.pt

1 | Tema do Trabalho

Este relatório versa sobre a resolução da segunda parte do trabalho da cadeira de Algoritmos e Estruturas de Dados na Faculdade de Engenharia da Universidade do Porto, realizado no primeiro semestre do ano letivo 2018/19, pelos alunos do grupo seis da turma três do segundo ano do Mestrado Integrado em Engenharia Informática e Computação: João Pedro Ribeiro (201707851), Rui Teixeira (201707001) e Vítor Ventuzelos (201706403).

O nosso trabalho tem como tema o tema quatro – Gestão de Bilhetes no Metro – dos temas sugeridos. O objetivo do trabalho foi resolver um problema recorrendo ao paradigma da programação orientada por objetos e utilizando a linguagem C++.

Na primeira parte do trabalho, o problema passava pela implementação de um sistema de gestão informática da bilhética e da utilização do Metro do Porto. Assim, era necessário processar uma série de informações sobre o metro (relativo a tarifas, tipos de assinatura e zonas de circulação) para fazer uma correta implementação da solução.

Nesta segunda parte do trabalho, tínhamos como objetivo implementar uma árvore binária de pesquisa onde se guardam as informações relativas aos funcionários do metro, uma fila de prioridade onde guardamos as composições existentes e uma tabela de dispersão com os utentes da empresa que já não compram bilhetes de tipo assinatura há um longo período.

2 | Solução Implementada

A solução implementada para resolver o problema proposto (sucintamente descrito na secção anterior) consistiu na implementação de três classes diferentes: a classe **Composição**, a classe **Funcionario** e a classe **BST**.

Na **main** acrescentamos às funções que respondiam aos problemas colocados na primeira parte do enunciado, uma série de funções que respondem aos problemas desta segunda parte do trabalho:

1. A função **gereFuncionarios** que permite ao utilizador do programa adicionar ou remover um funcionário ou imprimir a lista de funcionários da empresa, usando funcionalidades da classe Funcionario.
2. A função **gereComposicoes** que permite adicionar ou remover uma composição, imprimir a lista de composições da empresa, efetuar a manutenção de uma composição e decrementar o número de dias restantes para uma manutenção, usando funcionalidades da classe Composicao.

Para que seja possível a utilização destas funções na **main** são, obviamente, usados membros-função das classes **Composicao** e **Funcionario**.

Cada composição tem como elemento distintivo das outras composições da empresa o seu id\_composicao (do tipo int). A cada composição está associada ainda um booleano *avariado*, onde consta a informação sobre o estado de funcionamento da composição, e um inteiro diasProxManut, onde consta o registo do número de dias que faltam até à próxima manutenção da composição. A classe **Composicao** tem como membro-função funções básicas que retornam os dias que faltam para a manutenção seguinte (*getProxManut()*), o estado de funcionamento da composição (*getAvaria()*), o identificativo da composição (*getId()*), uma função que decrementa o número de dias que faltam para a manutenção da composição(*decDias()*), uma função que declara uma composição como estando avariada (*setAvaria()*), duas funções de comparação entre os dias que faltam para a manutenção (*operator <*) e de comparação entre o identificativo de duas composições (*operator==*) e, por fim, uma função que imprime as informações de uma determinada composição (*print()*).

Cada funcionário tem como elemento distintivo o seu salário (do tipo *int*), o seu nome (do tipo *string*) e o seu local de trabalho (um posto de venda). A classe **Funcionario** tem como membro-função funções que retornam o salário de um determinado funcionário (*getSalario()*), o nome de um funcionário (*getNome()*), o posto de venda onde o funcionário trabalha (*getLocalDeTrabalho()*) e funções que definem o salário de um determinado funcionário (*setSalario()*), o seu local de trabalho (*setLocalDeTrabalho()*), funções de comparação entre o salário de dois funcionários para efeito de ordenação (*operator<* e *operator==*).

3 | Lista de Casos de Utilização

Como casos de utilização possível para a solução proposta para a segunda parte deste problema, há, à semelhança daquilo que acontecia na primeira parte do trabalho, a perspetiva natural de utilização por parte da gestão do Metro do Porto.

Esta solução facilita a gestão da frota de composições (com a classe Composicao), fazendo um registo eficiente das composições disponíveis e da sua manutenção. Facilita ainda a gestão dos funcionários (através da classe Funcionario), com uma organização eficiente de vários dados relevantes sobre os funcionários e permitindo ainda a sua ordenação. Por fim, ajuda a perceber que clientes é que já não adquirem assinaturas há algum tempo, através da tabela de dispersão.

Apesar das especificidades deste programa para a resolução do problema específico da Metro do Porto, ele poderá ser facilmente adaptado para a utilização numa qualquer outra empresa para gestão de questões semelhantes.

6 | Principais dificuldades encontradas no desenvolvimento do trabalho

Não foram sentidas dificuldades no desenvolvimento da solução para o problema proposto.

7 | Esforço Dedicado

Esforço dedicado em termos percentuais por elemento do grupo:

* João Pedro Ribeiro: 30%
* Rui Teixeira: 15%
* Vítor Ventuzelos: 55%